Multi-model Analysis of Document Caches

Jeffery Baumes, Timothy Shead, Jason F. Shepherd and Brian Wylie

Abstract—The analysis of networked activities is dramatically more challenging than many traditional kinds of analysis. A network is defined by a set of entities (people, organizations, banks, computers, etc.) linked by various types of relationships. These entities and relationships are often uninteresting alone, and only become significant in aggregate. The analysis and visualization of the networks that are embedded in a document corpus is one of the driving factors behind the creation of our deep text analysis application named P2 (for Prototype 2). The development of this application utilized the open source Titan Toolkit. The toolkit's flexible, component based pipeline provides an excellent platform for constructing specific combinations of network algorithms and visualizations.

Index Terms—informatics; visualization; entities; extraction; graphs; networks; toolkit

1 Introduction

Our document analysis and clustering application is a stand-alone application to organize and classify documents based on their semantic content and perform named-entity extraction (people, places, etc.) to allow the exploration of the relationships between entities and documents. We refer to this application as "Prototype 2" or simply P2. P2 was developed to merge two normally distinct methods for text analysis into a single application. Document clustering uses statistical techniques such as latent semantic analysis (LSA) [8] to automatically detect topic clusters and classify documents under these topics. This provides a scalable, automated solution for obtaining a high-level view of the entire set of documents. We may also take advantage of the distance metric used in clustering to show the amount of similarity between any two documents in the dataset. The second technique is named-entity extraction. This enables enhanced searching through a corpus of documents based on content such as particular locations or people. In contrast to clustering, entity extraction provides an in-depth look into the content of individual documents, and can also link documents containing infrequent and interesting individuals or places. The challenge for P2 was to integrate information from both clustering and named entity extraction into a single useful application that incorporates the strength of each technique. Figure 11 provides a screenshot of P2.

Contributions This paper makes the following contributions to the research community:

- We demonstrate the flexibility of the Titan toolkit in performing text analytic processes.
- We demonstrate multi-model analysis of a document cache using a variety of text analytic tools.
- An application for displaying the results of the multi-model analysis is demonstrated.

This paper is organized as follows. After briefly reviewing previous work in this area in Section 2 we provide an overview of the Titan Toolkit in Section 3. We define multi-model analyses and discuss using datatypes and transformation in a pipeline environment to accomplish these analyses in Section 4 along with a discussion on the specific data analysis types and tasks performed. In Section 5, 6 and 7 we discuss the creation and setup of the pipeline and the interface presented to a user. In Section 8 results are examined and recommendations are made. Sections 9 and 10 present a detailed discussion of the user study conducted and the future directions for this work, respectively.

2 Background and Related Works

The field of network analysis and visualization is active and thriving. There are a large set of existing toolkits and frameworks that provide feature sets around network visualization and analysis capabilities.

- Prefuse Visualization Toolkit (www.prefuse.org):A Java-based toolkit for building interactive information visualization applications [13]. Prefuse provides an effective interaction and animation infrastructure for building many different types of views including network/graph diagrams.
- Tulip Toolkit (www.labri.fr/perso/auber/projects/tulip): A C++ toolkit for building interactive information visualization applications with both 2D and 3D display capability. Tulip implements some very fast algorithms for graph layout such as HDE [14]. Tulip also has a plug-in architecture for building and testing graph and tree layout algorithms.
- JUNG (jung.sourceforge.net): The Java Universal Network/Graph Framework is a software library that provides a common and extensible language for the modeling, analysis, and visualization of data that can be represented as a graph or network.
- GraphViz (www.graphviz.org): A set of libraries and executables, written in C, specifically for the visualization of many different types of graphs [12]. GraphViz mainly has functionality

- *Jeffery Baumes is with Kitware Inc., E-mail: jeff.baumes@kitware.com*
- *Timothy Shead is with Sandia National Laboratories, E-mail: tshead@sandia.gov*
- *Jason F. Shepherd is with Sandia National Laboratories, E-mail: jfsheph@sandia.gov*
- *Brian Wylie is with Sandia National Laboratories, E-mail: bnwylie@sandia.gov*

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

around static, non-interactive diagram generation. It has high quality graph layout algorithms and yields especially good results for directed acyclic graphs.

- InfoVis Toolkit (ivtk.sourceforge.net): A Java-based toolkit to ease the development of Information Visualization applications and components [10]. IVTK employs a unified data structure (a table of columns) for all of its graph, table and tree data in order to reduce memory footprint and reduce the complexity of filtering, selection and interaction algorithms. The InfoVis Toolkit is quite popular and allows the integration of several different types of views.

- InfoVis Cyberinfrastructure (iv.slis.indiana.edu/sw): Like Titan, this project provides an integration framework for other software packages. While the IVC framework itself is written in Java, it aims to allow contributors to integrate algorithms written in many different languages. The InfoVis Cyberinfrastructure uses the Eclipse Rich Client Platform for its application framework.

- Piccolo Toolkit (www.cs.umd.edu/hcil/piccolo): Piccolo is a layer built on top of optimized language-dependent graphics APIs. Currently supports Java and C# (Piccolo.Java, Piccolo.NET) [1]. Piccolo provides a built-in zoomable user interface (ZUI) that targets building applications where the user can transition smoothly from overview to fine detail.

- Pajek (pajek.imfm.si/doku.php) is a program, for Windows, for analysis and visualization of large networks.

While a majority of information visualization toolkits are written in Java, Titan is written in C++, allowing it to use a multitude of open source algorithm libraries, including the Boost Graph Library, Multi-Threaded Graph Library, Trilinos (Linear Algebra), CLAPACK (Linear Algebra), and Sandia-developed clustering and statistical Libraries including Matlab, R, and vtkStatistics [16].

3 Titan Overview

The Titan Toolkit [18] uses many of the visualization components in the open source Visualization Toolkit (VTK) and therefore shares the same component and pipeline execution model. This pipeline architecture allows algorithms and visualization to be flexibly combined in ways specifically focused on the application domain. In addition, the build system used by Titan (CMake [15]) allows it to be used on a wide variety of modern operating systems including Microsoft Windows, Macintosh OS X, Linux, and several variants of Unix.

In Titan, data processing components are called filters. Each filter implements a single algorithm as a C++ class. These filters may be connected together to form data processing pipelines. Figure 1 provides an overview of the stages of a typical pipeline. The beginning of a pipeline consists of data source filters that might read from databases or files. The output from a source filter is usually processed by filters that implement data transformations, algorithms, or other analysis techniques followed by a Titan view for visualization towards the end of the pipeline. Applications are usually centered on one or more visualizations and some UI that allows the user to change data sources, configure filter parameters, or change view settings.

The well-defined and documented APIs associated with the pipeline components make extending the toolkit straightforward. In most cases developers take existing filters that have similar interfaces to one that they wish to develop and use them as a template from which to build their new filter.

Titan also provides bindings for Python, Tcl, and Java that are automatically generated for every class. This provides the full functionality of Titan for applications written in these languages. These wrappers have proven themselves extremely useful for rapid prototyping of applications and experimental pipelines without the overhead associated with writing and debugging C++ code.

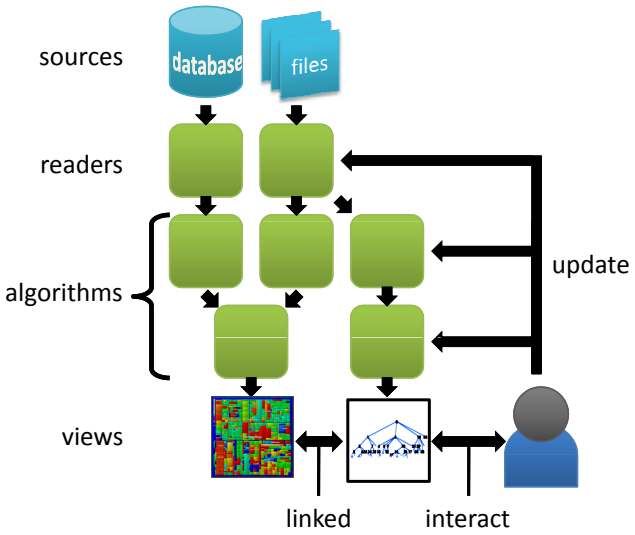

Fig. 1. Overview of the data flow model used by Titan.

3.0.1 Datatypes used by the Analysis Pipeline

The pipeline architecture used by Titan provides a core set of data structures that can be passed from component to component. End to end processing and analysis of data typically involves a comprehensive set of data structures and transformations.

- **Documents:** Clearly for our analysis of document caches we start with documents on disk (or in a database). Those documents are fed into the appropriate readers/drivers based on mime-type (MSWord, pdf, email, plain-text,etc). Most of the readers and database drivers in Titan produce tables.

- **Tables:** Tables (vtkTables) are the most general storage mechanism in the Titan toolkit and provide named columns of arbitrary types. In this case we use them to store meta data and text content of the documents to be analyzed, but in the general case they are used to store any type of entity attribute data. (example image here)

- **Graphs/Trees:** vtkGraph and it's various subclasses including vtkTree (see Figure 2) provide functionality around capturing relationships between entities. In additional to capturing topological relationships these data structures use vtkTables for attribute storage, this makes conversion from one form to another straight forward and efficient. An illustrative use case is described in the data transformation section below.

- **Sparse and Dense Matrices:** Tables, documents and other data structures can be converted to sparse or dense matrix formats to enable linear algebra to be used to delve more deeply into the data.

The hierarchy of graph data structures is shown in Figure 2. Blue classes represent the primary data structures that are passed between filters as pipeline objects. The subclasses of vtkGraph provide specialization for graph types such as undirected vs. directed graphs and trees, etc.

Titan gains several advantages from the use of a class hierarchy for graph types. Filters that operate on a vtkGraph will work for all graph types. The use of subclasses also allows filters specific to particular graph types to be developed, i.e., a filter taking only a directed graph as input would fail if it was given an undirected graph.

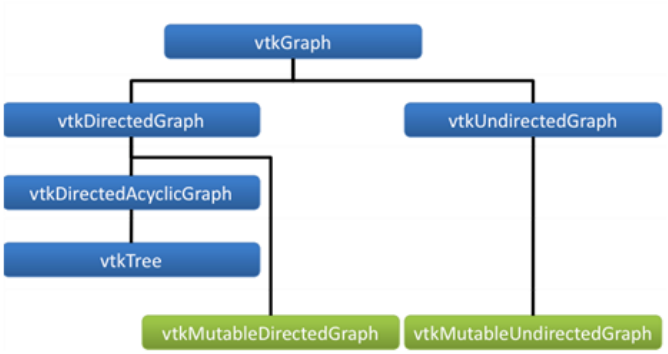

Fig. 2. Hierarchy of Titan graph classes.

The Titan graph data structures in blue also enforce the criterion that the data must be valid at all times. The separate, mutable classes (shown in green) help enforce this criteria in that they are the only graph classes which contain methods for adding or removing vertices and edges. This is necessary as a precaution since it is difficult to maintain a valid data structure for some graph types during construction. In particular, this can occur with the construction of a tree - it is possible at an intermediate step during construction to have a graph that is not a valid tree. By providing specialized mutable graph classes for construction along with methods to safely convert them into more specialized types, we ensure the validity of the data structures at all times in the pipeline, while avoiding the excessive overhead from runtime validation.

3.0.2 Data Transformations

As you look at the numerous data types utilized in an analysis pipeline it become apparent that data transformations play a central role in the end-to-end process. The efficient transformation of data from one form to another is a key element of a functional, usable analysis toolkit.

Tables to Graphs: Typically an analyst is not only interested in entities and attributes but also in the relationships and links between entities. When these links are stored as tabular data (in a database or flat files) we must provide a way to flexibly extract the entity relationships. In Titan we implement this using the vtkTableToGraph filter.

To illustrate lets use the analysis of employee data; we have data on employees, the employee's publications and data on emails between employees. We hit the database with queries like 'select * from employee-table', 'select * from employee-pubs', and 'select * from email-march-2010', using the toolkit database drivers these queries are returned as vtkTables. Figure 3 shows up the ways we might want to construct relationships

- a) Email (From-To) would show us the email transaction network between employees.
- b) Employee to Company would show us the relationship between employees and the companies they work for.
- c) Employee/Author to Article to Topic would give us a 'publication map' of which employees are collaborating and in which areas.

You run For more details on this process see [18].

Tables to Trees: For the transformation of a table into a tree we use two filters, vtkTableToTree and vtkGroupLeafNodes. vtkTableToTree simply takes a table and creates a single rooted tree with the table elements as leaves. Once all the data exist in this trivial tree structure we use vtkGroupLeafNodes to organize it into an arbitrarily leveled tree based on the series of vtkGroupLeafNodes filters used. This allows us to organize our employee publication data in another form for analysis, processing and display Figure 4.

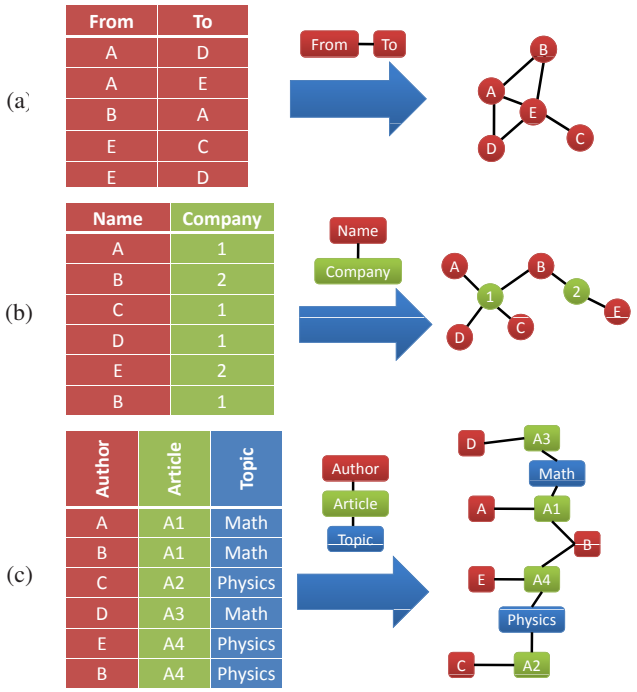

Fig. 3. How vtkTableToGraph translates an edge attribute table (left) into graph edges (right) based on a link graph (center).

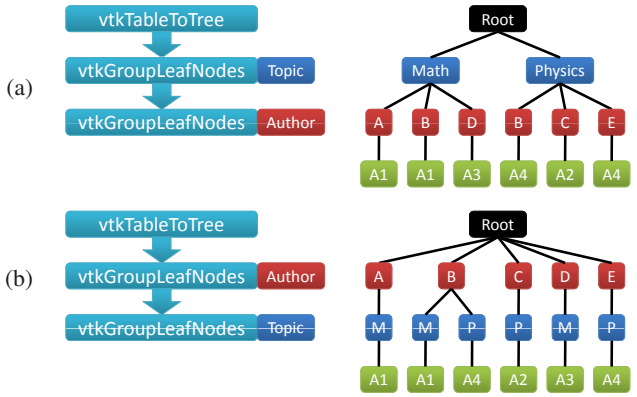

Fig. 4. vtkTableToTree and vtkGroupLeafNodes used to convert our employee publication data into a tree.

Tables to DenseArrays: A table can also be viewed as a simple dense array implementation. Specific methods for converting tabular columns to columns in a matrix are available and can be used in a memory efficient manner to avoid deep copies of the data to convert between formats.

Graphs to SparseArrays: A alternate representation of a graph is also a matrix. Methods are available for converting graphs to sparse arrays (for memory efficiency) and from sparse arrays to dense arrays. Sparse array solvers (including highly-parallel solvers) are also available for enabling a wide range of linear algebra methods to be applied to these matrices.

4 Multi-model analysis

Multi-model analysis involves using several types of modeling techniques with the data and presenting the results in multiple views, hybrid or combined formats. P2 utilizes several types of text analyses. In this section, we outline many of them, including Latent Semantic Analysis, Entity Extraction, Clustering, and contained graph searches.

The coupling of these analyses is a powerful technique and provides greater access and insight into the document cache.

4.1 Latent Semantic Analysis

Latent Semantic Analysis (LSA) is a natural language processing technique used to analyze relationships between a set of documents and the terms contained in the documents. This is accomplished by generating sets of topics related to the documents and the terms. LSA utilizes a term-document matrix, or frequency matrix, where rows correspond to terms found in the document cache and columns represent the documents. Each value in the matrix is proportional to the frequency of a term within the document (in an unweighted matrix each value would be equivalent to the frequency of a term within a document).

Following the construction of the frequency matrix, the LSA method finds a low-rank approximation to the term-document matrix. The result of this rank-lowering is that several dimensions will be combined to depend on more than a single term, mitigating the problems of identifying synonymy[1], and developing topical clustering in the high dimensional space. Rank-lowering also partially mitigates the problem of polysemy[2] by weighting terms that commonly occur together in similar dimensions.

Rank-lowering can be accomplished in one of many ways, although the preferred method is to utilize a Singular Value Decomposition (SVD) of the frequency matrix. By selecting the k largest singular values from this decomposition, along with the singular vectors, you can get the rank k approximation of the original frequency matrix with the smallest error. Each term and document vector in the original matrix can be treated as residing in a topical space of k dimensions, and we can evaluate documents in relation to each other by calculating distances between documents in this k-dimensional space.

4.2 Clustering

Clustering in P2 is accomplished using the GMeans[7, 4, 6, 5] set of clustering algorithms. The GMeans package has been incorporated as a single filter within the Titan set of tools, and exposes all of the options available from GMeans traditional command-line interface.

GMeans is a tool that generates a k-means style of clustering, incorporating four different distance or similarity measurement options and several initialization methods to generate a hard clustering. Similarity options for clustering include cosine or spherical k-means, Euclidian distance, Kullback-Leibler divergence, and diametric distance. The P2 pipeline hard-codes the use of spherical k-means to calculate cluster distances.

The input matrix to the GMeans filter may be of a dense or sparse-type along with a target number of clusters desired (i.e., k). The output is an array consisting of a cluster id for each of the documents in the cache.

4.3 Entity Extraction

Entity extraction is the process of classifying and sorting textual elements of a document into predefined categories, most typically persons, locations, organizations, quantities, etc. Entity extraction is also known as entity identification and named entity recognition (NER).

Entity extraction algorithms are commonly of two main algorithm types: grammar or rule-based techniques and statistical models. The grammar or rule-based techniques commonly out-perform the statistical models, but at the cost of substantial increases in development and overhead by language experts. Statistical models, on the other hand, must be trained using manually annotated training datasets (the larger the training set, the more accurate the results). Training in a given domain, or document collection, does not always translate well into a new domain or with a new set of documents. This is true for both grammar based and statistically based entity extraction techniques; therefore, tuning the extraction algorithms is beneficial for all new data.

Entity extraction in P2 is accomplished using the Stanford Named Entity Recognition (SNER) tools. SNER is a statistical model based entity extraction tools that utilizes a conditional random field (CRF) for labeling and parsing of sequential data. A conditional random field is an undirected graphical model where each vertex (word) represents a random variable whose distribution will be inferred and each edge represets a dependency between two random variables. The parameters of the distributions must be learned, and the basic assumption in training a CRF is that some variables are always observed and others variables will be inferred making it possible to train the model to maximize the conditional probability between observed and inferred variables. A CRF model can also be viewed as a Markov random field that has been trained discriminatively.

Titan provides access to SNER via a single filter which takes a table of input documents, with one document per row in the table. The output is also a table containing a set of addresses into the document collection of each entity identified, along with a tag identifying the type of entity. The entity extraction model can be individualized for each document's mime-type, and standard models are provided for common mime-types.

4.4 Contained Graph Search

Titan provides access to a variety of graph algorithms and heuristics that are useful for the analysis of network data. A set of adapter classes are provided to enable compatibility with third party graph libraries including:

- Boost Graph Library (BGL)
- Parallel Boost Graph Library (PBGL)
- Multi-Threaded Graph Library (MTGL)

The fusion of multiple libraries into the pipeline architecture allows application developers to apply the best features of each for a problem of interest. Further, Titan provides adapters with shallow-copy semantics that minimize copying when crossing library boundaries, an important consideration when working with large datasets. Table 1 provides a listing of available graph algorithms and their source libraries[3].

While Titan does not yet wrap the full set of algorithms provided by these libraries we are incorporating additional capabilities continually. Over time, Titan will provide pipeline components for most of the algorithms provided by these libraries.

Table 1. Titan graph algorithms provided.

Algorithm	*BGL*	*MTGL*	*PBGL*
Breadth-First Search (BFS)	X	X	X
Biconnected Components	X		
Connected Components	X	X	X
Connection Subgraphs		X	
Minimum Spanning Tree (MST)	X		X
Network Centrality	X		
R-MAT Generator		X	
Single Source Shortest Path		X	
S-T Shortest Paths		X	X
Subgraph Isomorphism		X	
wCNM Community Detection[2]		X	

[1] synonymy - different words have similar meaning or convey the same ideas, for instance, 'physician' and 'doctor'

[2] polysemy - same word conveys different meanings, for instance, the term 'bed' has a different meaning in each of these phrases: 'river bed' versus 'sleeping bed'

[3] MTGL Subgraph Isomorphism implements an inexact heuristic for attributed-graphs only.

```
from vtk import *
from titan.MTGLGraphAnalysis import *

# create a random graph
src = vtkRandomGraphSource()

# create MTGL CC filter
cc = vtkMTGLConnectedComponents()
cc.SetInputConnection(src.GetOutputPort())

# call BGL biconnected components
bicc = vtkBoostBiconnectedComponents()
bicc.SetInputConnection(cc.GetOutputPort())

# send graph to visualization...
view = vtkGraphLayoutView()
view.SetInputConnection(bicc.GetOutputPort())
```

Fig. 5. Sample Titan pipeline created using Python.

5 Scripting with Titan

Titan provides automatic wrapping of every class enabling access to pipeline filters from languages such as Python, Java, and Tcl. We believe this feature is very useful, particularly as a prototyping tool when developing data pipelines and full-fledged applications. Figure 5 demonstrates a sample Python script that creates a random graph and then executes the connected components algorithm from MTGL and the biconnected components algorithm from BGL. This example demonstrates the flexibility of the pipeline architecture since we are able to run graph algorithms from both BGL and MTGL on the same graph without having to deal with data compatibility issues between these libraries. Titan provides the adapters for these libraries enabling seamless integration.

6 The pipeline

For this task we leveraged the breadth of the toolkit to construct this domain specific application.

- Database Drivers
- Web Server/Client tiers in Titan
- Latent Semantic Analysis (LSA) for text analysis
- Document Clustering
- Statistics Components
- Graph Algorithms
- Protovis [3] (for presentation of results in a web browser)

While a detailed description of all of the functionality is beyond the scope of this paper we will cover the components and their usage to address the use case. We begin the process by pulling the data from the database with the database classes in the toolkit. Database query results are returned as tables, which are then used to generate a graph. Once we have the data in the form of a graph we then leverage several of the graph algorithms mentioned above in addition to the weighted Clauset-Newman-Moore (wCNM) [2] community detection algorithms. At a high level, the wCNM algorithm identifies communities by clustering vertices so that the edge density within a cluster is much higher than the edge density between clusters. When this clustering process finishes, each cluster is labeled as a community.

After identifying web communities, we use Titan components to conduct textual analysis (LSA) on the raw web page text directly. This process identifies significant topics and concepts within the database of web pages. We extract topic scores for each web page and use those as a basis for further document clustering. We also compute the entropy of each term within the vocabulary in order to identify terms of likely interest. The results of this analysis are are then finally presented to the analyst as a web page consumable by any modern browser. Clearly a significant amount of complexity in the text analysis process. Figures 8,9, and 10 illustrate the components in use and how they are connected in the pipeline based toolkit.

7 The User Interface

As we stated in the introduction an algorithm is only truly useful when it can be deployed as part of a system. Here, we present two applications where we applied the Titan data structures and algorithms. The first addresses a document-analysis use case. The second is intended for exploration and analysis application and the second example demonstrates an application that analyzes a database of web pages obtained using a crawler.

After loading a corpus of documents, P2 computes document-to-document similarities using LSA. This information is used to produce a graph where individual documents are represented by nodes, and are linked to other nodes when the document similarity exceeds a threshold. Documents are also clustered into groups containing documents that are similar to each other. Finally, we display the similarity graph to the user in a tree-ring view as shown on the bottom-left corner of the application window. The hierarchical nature of the tree-ring view allows a large quantity of information to be conveyed in a compact, visually appealing manner. The outer ring represents clusters and the inner ring represents documents within each cluster. The interior of the ring contains edges connecting similar documents. This view enables the user to identify documents that are highly similar to documents in other clusters.

In addition to document-to-document similarity, we also extract named entities such as names, places, organizations, etc. from the content of the documents using the Stanford Named Entity Recognition tool (StanfordNER) [11]. StanfordNER is written entirely in Java, which posed a challenge to the C++-based P2 application. To overcome this, we created a generic means to call Java components from within the C++ pipeline using JNI (Java Native Interface). As a result of this work, any Java component may be integrated into a Titan pipeline.

Using the named-entity information, we generate a document-entity co-occurrence graph where nodes are documents or entities and entities are nodes and edges connect each entity to the documents in which they appear. We then merge the document-similarity and document-entity graphs together to produce a new graph containing both entity-to-document relationships and document-to-document similarity relationships.

The central view focuses on a currently selected document. The current selection may come from any of the views on the left or right of the application. This view uses Qt's built-in WebKit view, which provides web-browser capabilities including standards-based HTML document layout and CSS styling. The entities detected in the current document are highlighted and may be hyperlinked to an appropriate database to show additional information about the person, place, or organization. Figure 6 shows an example of linking entities to the corresponding page on Wikipedia.

Users are able to select specific entities that are interesting by dragging them into the hotlist, which appears just below the list of named entities. P2 then computes paths between entities, including the documents through which they are connected, that are interesting using the connection subgraphs [9] algorithm. The connection subgraphs algorithm attempts to find short, direct paths between nodes in a graph by penalizing long paths as well as paths which pass through high-degree nodes. This algorithm is more useful than a simple shortest-paths algorithm when analyzing small-world graphs (such as those arising from social networks) thanks to the penalty it assesses against high-degree nodes. In a social network, for example, a high-degree node may represent a person such as a politician who has made a very large number of very brief contacts with other people. Few of this person's contacts would be useful in describing substantive connections between two entities. By penalizing such high-degree nodes, the connection

```
# open database...
database =
vtkSQLDatabase.CreateFromURL("postgres://blah)
database.Open("")

# setup query...
query = database.GetQueryInstance()
query.SetQuery("select source, target from
weblinks")

# populate table with query...
table = vtkRowQueryToTable()
table.SetQuery(query)

# convert table to graph...
graph = vtkTableToGraph()
graph.AddInputConnection(table.GetOutputPort())

# send graph to visualization...
View = vtkGraphLayoutView()
View.SetInputConnection(graph.GetOutputPort())
```

Fig. 6. Python psuedo-code performing a query against a database, storing the results in a table, converting the data into a graph and displaying the results (see Figure 11 for the results of this process, Figure 7 is generated by running the graph algorithms in Figure 5 and switching the view to a vtkTreeRingView).

Fig. 7. Detailed view named-entity/document similarity graph with connection subgraphs algorithm linking entities together.

subgraphs algorithm is in effect trying to find connections between people and entities that have a greater likelihood of being important.

Figure 7 shows the subgraph generated by computing the connection subgraphs between each entity in the hotlist. The result is a small graph showing connections between entities that the user is interested in and the documents in the corpus that relate them to each other.

P2 represents an early success of the Titan Toolkit to deliver an end-to-end application to analysts for use in a production capacity. The flexible pipeline architecture and component based nature of Titan enabled our team of developers to produce P2 within a short time period. The ability to take a corpus of documents and automatically organize them into groups containing similar topics, extract the named-entities from them, and then link documents to entities in a searchable manner has been well received by our analyst community. The dataflow pipeline used to generate the application is detailed in Figures 8,9, and 10 and a screenshot of the final application is shown in Figure 11.

8 Results

The P2 application was developed as a research prototype to test functionality developed as part of a Sandia National Laboratories Lab-Directed Research and Development (LDRD) program entitled the "Networks Grand Challenge" (NGC). In terms of a prototypical application to test and expose an analytic community to raw research being conducted in the Labs' research community, P2 was a major success. This tool introduced the capabilities and talents to the analysis community in a way that has whetted appetites and bolstered communication between the research community and the analysis community that will engender dialogs and cross-collaborations for many years to come.

8.1 User-Studies

Shortly after initial delivery, a brief study was conducted by Stubblefield[17] to evaluate capabilities provided by the P2 application, the interaction space the application defines and users' activities and experiences in using the tool. General concensus among analysts confirmed that the P2 application had *high potential value*, but deficiencies in design and engineering were needed to improve the tool's utility. This is to be expected of a tool designed to prototype capabilities, and follow-on tools should take into account specific findings of the report. Specific areas of concern were in tool-to-tool interactivity (exporting findings in formats that can be evaluated within other tools), providing for improved user interaction (i.e., allowing the user to interact more heavily with the graph-based views), improved clustering and user-guided clustering, additional statistically-based calculation capabilities, and better support for iterative queries and processing.

9 Conclusions

We demonstrated the use of the Titan toolkit within the P2 application showing the utility of integrating graph algorithms with text analysis techniques to organize and analyze large corpuses of documents. The utility of these algorithms and proper visualization techniques is also shown in our web crawl data analysis in which we used visualization techniques in our process of verifying the structure of a large incoming data set.

The flexible pipeline architecture also eases the burden on application development somewhat due to the uniform nature in which algorithms are combined together into complex pipelines. As demonstrated, rather complex data pipelines can be assembled to build end-to-end applications that solve difficult problems.

Support for additional languages such as Python enables rapid prototyping of experimental pipelines without the overhead of developing C++ code which can save time and developer resources.

10 Future Works

We recognize that a toolkit does not succeed without community support and contribution. We are actively seeking members of the community to collaborate with us on Titan 2.0 as users, contributors and developers. Further documentation and checkout instructions for the Titan toolkit can be found at http://titan.sandia.gov.

Acknowledgments

The authors would like to thank the extended "Titan Family": Patricia Crossno, Marcus Hanwell, Berk Geveci, John Greenfield, John Harger, Kenneth Moreland, Thomas Otahal, Dave Partyka, Philippe Pebay, David Rogers, Eric Stanton, and David Thompson.

REFERENCES

[1] B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering*, 30:535–546, 2004.

[2] J. W. Berry, B. Hendrickson, R. A. Laviolette, and C. A. Phillips. Tolerating the community detection resolution limit with edge weighting. *arXiv:0903.1072v2[physics.soc-ph]*, Mar. 2009.

[3] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, pages 1121–1128, Nov/Dec 2009.

[4] I. S. Dhillon, J. Fan, and Y. Guan. Efficient clustering of very large document collections. In V. K. R. Grossman, C. Kamath and R. Namburu, editors, *Data Mining for Scientific and Engineering Applications*, pages 357–381. Kluwer Academic Publishers, 2001. Invited book chapter.

[5] I. S. Dhillon and Y. Guan. Clustering large and sparse co-occurrence data. In *Proceedings of the Workshop on Clustering High-Dimensional Data and its Applications at the Third SIAM International Conference on Data Mining*, 2003.

[6] I. S. Dhillon, Y. Guan, and J. Kogan. Iterative clustering of high dimensional text data augmented by local search. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, 2002.

[7] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, Jan 2001.

[8] D. M. Dunlavy, T. M. Shead, and E. T. Stanton. Paratext: Scalable text modeling and analysis. *ACM International Symposium on High Performance Distributed Computing*, June 2010.

[9] C. Faloutsos, K. S. McCurley, and Tomkins. Fast discovery of connection subgraphs. *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*, 2004.

[10] J. D. Fekete. The infovis toolkit. *10th IEEE Symposium on Information Visualization (InfoVis'04)*, pages 167–174, 2004.

[11] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL2005)*, pages 363–370, 2005.

[12] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. A technique for drawing directed graphs. *Software Engineering*, 19(3):214–230, 1993.

[13] J. Heer, S. K. Card, and J. A. Landay. Prefuse: A toolkit for interactive information visualization. *ACM Human Factors in Computing Systems (CHI)*, pages 421–430, 2005.

[14] Y. Koren and D. Harel. Graph drawing by high-dimensional embedding. *LNCS Graph Drawing (GD'02)*, 2528, 2002.

[15] K. Martin and B. Hoffman. *Mastering CMake*. Kitware, Inc., 2008.

[16] P. Pebay. Statistics. *The VTK User's Guide, ISBN: 978-1-930934-23-B, Kitware, Inc.*, 11th ed.:192–198, 2010.

[17] W. Stubblefield. An evaluation of the network grand challenge p2 prototype from an interaction design perspective. *SAND Report*, August 24, 2010.

[18] B. Wylie and J. Baumes. A unified toolkit for information and scientific visualization. *IS&T/SPIE Electronic Imaging, Visual Data Analytics*, 2009.

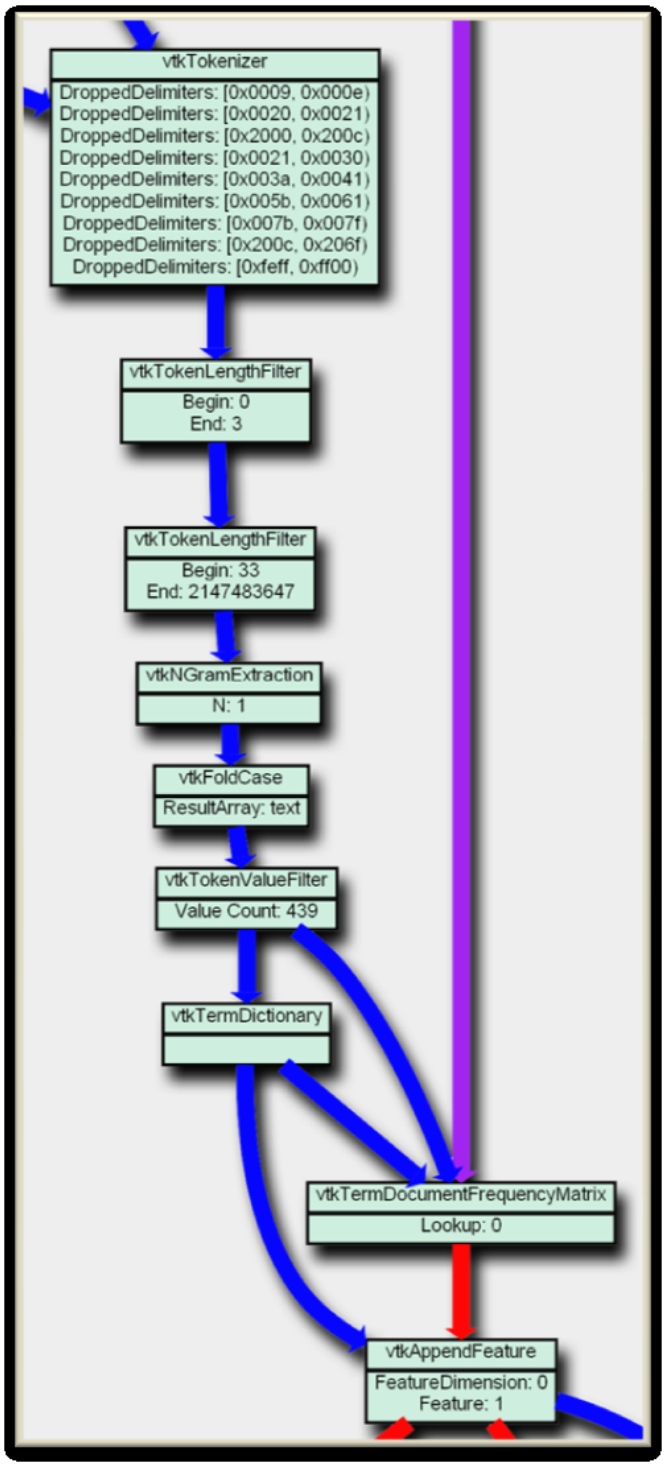

Fig. 8. Part one of the text processing pipeline; tokenization, case folding, term dictionary and term frequency matrix creation. This pipeline (or a variant of it) is used for both the P2 application and the web crawler application.

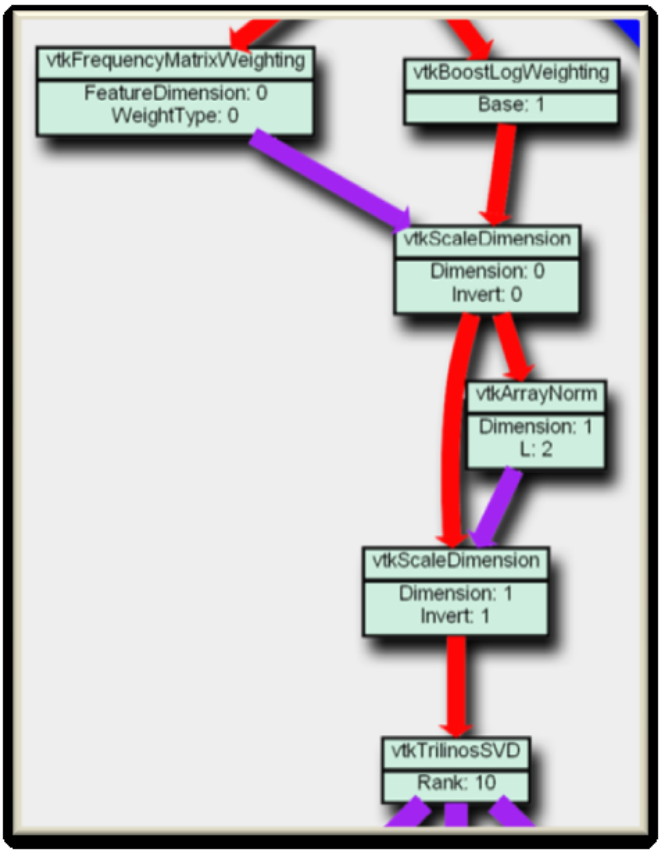

Fig. 9. Part two of the text processing pipeline; document features are weighted, scaled, normalized, and scaled again in preparation for the Trilinos Singular Value Decomposition (SVD) sparse matrix calculation.

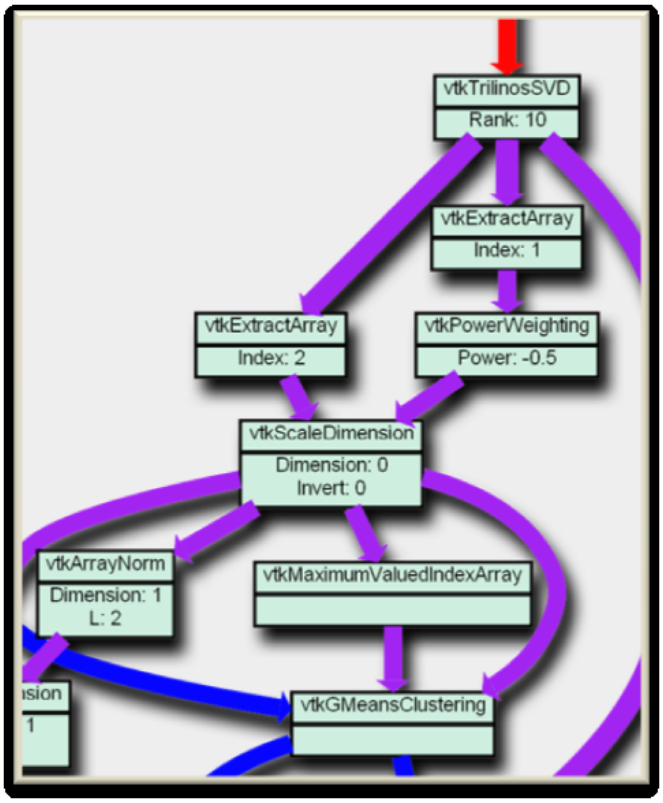

Fig. 10. Part three of the text processing pipeline; the SVD input is a sparse term/document matrix, the output is a set of three dense matrices: right singular vectors (term vs features), singular values, and left singular vectors (terms vs concepts). The matrices are then weighted, scaled and sent to the document clustering filter which groups documents in concept space.

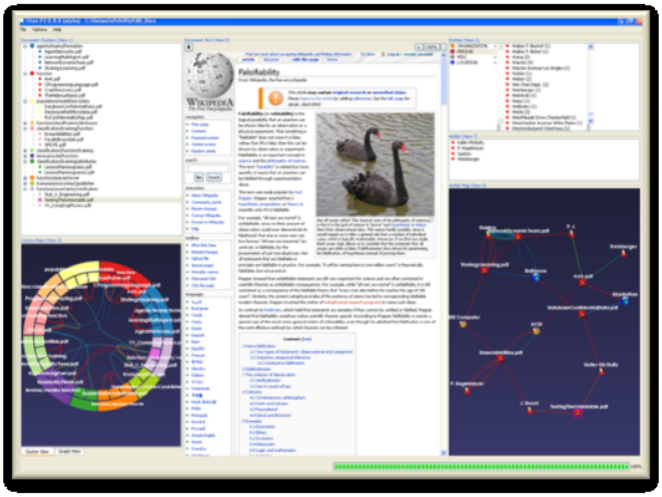

Fig. 11. Document analysis and clustering application screenshot.